Flexible Shape Models for Marine Animal Detection in Underwater Images

by

Anthony Francis Kelsall, BComp

A dissertation submitted to the
School of Computing
in partial fulfilment of the requirements for the degree of

Bachelor of Computing with Honours

University of Tasmania

(November, 2005)

Declaration

I, Anthony Kelsall, declare that this thesis does not contain any material which has been accepted for the award of any other degree or diploma in any tertiary institution. To my knowledge and belief, this thesis does not contain any material previously published or written by another person except where due reference is made in the text of the thesis.

Anthony Kelsall

Abstract

Many industries are benefiting from computer automation, however the area of image analysis is still limited. The process of finding a potential object in an image is hard in itself, let alone classifying it. Automating these tasks would significantly reduce the time it takes to complete them thus allowing much more data to be processed. This becomes a problem when data is collect faster than it can be analysed. Images and video sequences are captured for different purposes and need to be manually processed in order to discover their contents.

The fishing industry is a perfect example of this. A fish farm needs to know the average size of the fish in a ring. At present, this involves either manually taking a sample of fish from the ring and measuring them, or taking a series of stereoscopic images and manually tracing a sample of fish. By using active shape models, the process of tracing a fish sample can be automated. The Active Shape Model (ASM) Toolkit is an implementation of active appearance models, an advanced type of active shape model. The wrapper application that was written as part of this research allows a more streamlined process to input region data into the ASM Toolkit for searching. Once a sample has been matched, it is possible to use the key points around it to base further calculations on such as its size and weight.

The ASM Toolkit and the wrapper program demonstrate how the process of identifying a fish in an image can be automated and that it is possible to calculate the size and weight of fish. In an ideal manual test, the most effective model matched 68% of samples, and in the automated test matched 50% of the samples. If the program can run over several days collecting appropriate samples, the model will be able to match enough fish to estimate the average size and weight within a ring. It is shown that the types of samples used in training the model affects the performance more than the number of samples used.

Acknowledgements

Firstly I would like to thank my supervisor Dr Ray Williams for the amazing support and guidance throughout the year. I very much appreciate the amount of time and effort put into helping me prepare this thesis. Also I would like to thank Dr Tim Pouly and AQ1 Systems for providing the images used in this research.

Thanks to my friends outside of honours for keeping me relaxed, guiding me and supporting me this year. Special thanks to Rob for looking out for me and always encouraging me. Special thanks also to James for being one of my best friends and also for coming back to Tassie! Thanks to Kat, Venetia and Rosie for being there for me and keeping me relaxed and happy.

A huge thanks to all the honours guys for so many fun, amusing and interesting hours spent at Uni. Particularly to David Nahs sitting behind me, Michael, Ivan, Tristan (Lambert), Tristan (Ling), Emma, Dave (Burella), Dima, Simon, Duncan, Hamid, Bruno, Steve and Hallzy. Also to Joel the PHD student for hanging around.

Thanks to all my friends from tennis for always having so much fun. Thanks to Anna for getting me up early in the final few days and hitting with me. To Larn for being a great doubles partner this year. To James for our quality hits and helping me improve. To Ella for being so happy and Josh for keeping it real.

A special thanks to Louisa and Ellie for providing me with 45 minutes per week to totally forget about everything to do with Uni. For being so happy, interesting to talk to and well behaved. Thanks also to their parents for being so flexible and understanding and always trying to fit in with my busy schedule.

Finally, I would like to thank my family for supporting me throughout the year. Thanks to my mum for trying to get me out of bed so I could spend at least a few hours a day working at Uni. Thanks to Dad for asking me so many computer questions and the amazing proof reading at the end! And thanks to my sister Diana for watching “Friends” with me, proof reading and correcting my entire thesis for me!

Contents

Figures and Tables

List of Figures

List of Tables

Chapter 1

Introduction

In Australia, many industries are using the ocean and its resources for various purposes. As a result there is a growing interest in analysing underwater images and image sequences using computers. At present, much of the processing is carried out manually requiring a significant amount of time and resources. The automation of such tasks would allow the industry to process much more data with less human interaction. There are many possibilities for automation of simple tasks and a growing realisation of the more complex tasks such as identifying specific objects in images and classifying them.

Marine researchers are already finding that the newly developed sensing techniques are amassing data far more rapidly than can be analysed manually and so demand for automated techniques is growing. Being able to automatically locate, identify and track marine animals in video sequences would enable automated inventory of various marine species to be carried out (Dirk, Edgington & Koch 2004). The automated approach would also be able to identify and isolate certain footage of particular objects or events. This would greatly reduce the amount of footage that scientists have to analyse (Edgington et al. 2003). In the near future, automated tracking of individual animals by autonomous underwater vehicles will increase the amount of data that can be collected (Dirk, Edgington & Koch 2004) and so event analysis will become more important.

Chapter 2

Literature Review

This chapter will describe the digital imaging area and give a background into the techniques. This will be followed by a justification of why automation is necessary and an overview of the techniques used to achieve it. Lastly, a detailed description will be given of the process of building an active shape model and searching for a sample.

2.1 Computer Imaging

Image analysis and processing is a field that has been worked on for a number of years. It continues to gain momentum with new applications being researched and developed. Computer imaging can be divided into two distinct areas. One of these is computer vision where the output images are primarily for the computer to use and analyse. The other area is image processing where the output from the applications is for human consumption (Umbaugh 2005).

Image analysis falls under the computer imaging umbrella and involves examining the data from the image for a specific application (Umbaugh 2005). The process of image analysis can include image segmentation, image transforms, feature extraction, and pattern classification (Umbaugh 2005). Image segmentation is one of the first steps taken when finding higher-level objects in the actual image data. Feature extraction is a process that is used to find higher-level information about an

image such as any shapes that may appear or colour information. This feature extraction process may require the use of image transforms to find and identify the spatial frequency information needed. Lastly, pattern classification is the process by which higher-level information from previous processes is used to find objects within the image (Umbaugh 2005).

2.2 Marine Science by Video

The ocean is a very important resource and is gaining more attention as certain industries are having a negative impact on it. Whaling, fishing and pollution are affecting the abundance of fish and harming the environment. More and more studies are being undertaken to determine exactly what is causing the most harm and where.

Previously, the method used to study organism diversity, population, and the distributions within an area was essentially destructive. It consisted of dragging a net behind a boat and physically collecting the samples. This method only gave an indication of the area of interest because it combined the results over the entire drag area (Wilson 2003). This made it impossible to extrapolate the findings to the surrounding areas.

Video footage can be used to collect a large amount of image data from an area. This data is then analysed and the key objects and events are extracted. At the Monterey Bay Aquarium Research Institute (MBARI) in Moss Landing, several remotely operated underwater vessels collect footage for the scientists to analyse. To study an environment or species, many hours of detailed footage is collected, often at very high resolutions to get the maximum detail in the images.

The sheer volume being collected is becoming too much for scientists to analyse using current methods. At MBARI, they take on average four hours to identify the organisms in a single video (Wilson 2003). During each dive, six to seven videos are collected which is a massive amount for any laboratory to process. Over 14,000 tapes have been collected containing more than 10,000 hours of footage (Wilson 2003).

2.3 Marine Applications using Automated Image Analysis

Techniques are being developed using computers to assist in the analysis of these images to significantly reduce the amount of manually processed data. The ultimate goal of such a system is to autonomously detect an object, identify it and classify it within a database of known objects in real time. In order to achieve this, an accurate and efficient algorithm will need to be used. Several research papers have studied techniques for identifying animals in underwater images. They have approached the problem from two different angles.

Tillett et al (2000) described a semi-automated process aimed at calculating the physical characteristics of an underwater fish, such as its size and weight. In order to calculate the dimensions of the fish, two cameras were set up close to each other to determine the distance of the object from the camera stereographically. The process still requires a human to identify the key points on the fish in order to apply the model. While this certainly does reduce the time required to calculate the characteristics, it is not an ideal solution.

Wilson (2003) addressed the problem of actually identifying marine animals in the video sequences and determining their species. They described a fully automated approach where the analysis was done in real time as the footage was recorded. This meant that identification of very small indistinguishable objects was extremely difficult, as it required significantly more processing. They acknowledge that because of the nature of the objects being detected and their similarities, complete trainability of the system would be difficult.

Another application of computer vision is to provide automated navigation and tracking capabilities for Autonomous Underwater Vehicles (AUVs) being used to record the underwater footage. Such vehicles use vision systems employing sonar detectors and cameras to identify their surroundings and objects in their close vicinity. Another technique used to to study a marine species in detail, involves scientists manoeuvring Remotely Operated Vehicles (ROV) to follow the animal underwater. This requires continuous hours of precise movements and corrections by

the operator to keep the animal in the image frame. This becomes very stressful and pilot fatigue is a big problem (Rife & Rock 2001). As a result, lengthy observations cannot be conducted due to human limitations. However, with the use of an AUV, this task could theoretically be carried out indefinitely, allowing scientists to observe a species over a significant period of time.

2.4 Object Identification Techniques

When referring to computer vision, the ultimate goal is image understanding. A computer can be programmed to analyse an image in various ways with different goals in mind. Image understanding is a broader picture which aims not only to delineate the individual parts of an image but to identify the objects they represent (Cootes, T. F. & Taylor 2004).

There are two approaches to identifying objects in an image. The first is the 'bottom-up' approach where the raw data of the image is analysed searching for edges and regions. A region is made up of connected edges and might represent an object or area of interest within an image (Ballard & Brown 1982). Any features identified are assembled into groups in an attempt to discover what they are. The problem is that the computer has no knowledge of what it is looking for. As a result, this approach is often error prone and harder to adapt to changing conditions and object variations (Cootes, T. F. & Taylor 2004).

The other approach is the 'top-down' approach. Rather than examining the raw data, the computer searches the image for a general shape that it has been told to look for. This shape is known by the computer prior to the searching process and is used to find the most likely match within the image being analysed. Once it has found a match, further calculations can then be carried out to identify whether it is in fact an instance of the desired shape (Cootes, T. F. & Taylor 2004). An example of a 'top-down' approach is the Active Shape Model (ASM) approach, which will be the focus of this review.

2.5 Active Shape Models

The first incarnation of adaptable models was in the form of an active contour model. They are also known as snake models because they incorporate a spline curve that aligns itself with the edges found in the image. The curve parameters are weighted in such a way that certain bends are not possible. They have features such as stiffness and elasticity and are parameterised which allows points to be controlled (Kass, Witkin & Terzopolous 1987). These work for ideal objects but still rely on the 'bottom-up' approach dealing with raw data.

An extension of the contour model assigns a weight to the splines representing a preferred home location for the particular point. This allows the model to have a default shape that can then be adapted according to variations in the detected edges (Hinton, Williams & Revow 1992). This does not however, cater for variations very well as it only roughly defines the possible variations depending on the number of control points (Cootes, T. F.Taylor et al. 1994).

A large number of papers have been written focusing on the development of active shape models. These models have been developed mainly for facial recognition (Cootes, T. F., Edwards & Taylor 1998) and medical imaging (Cootes, T. F.Hill et al. 1994). These are complex applications with many subtle differences in the appearance of the shapes and features. There has also been a study into the suitability of active shape models for use with marine starfish (Kuksin 2001).

Active shape models are based on the same principals as contour models, providing a flexible means to identify objects within an image. Each model is a collection of labelled points defining the boundaries of a specified shape. Using a training set of images, the computer extends the model by obtaining statistics of variations between the points. Using the mean value of each point, a model representing the average appearance in the training set is obtained. The main object deformations are also known giving a number of modes of variation describing the ways the object in the training set tends to deform from the mean. This produces a point distribution model

with a number of parameters that can be altered during a search to identify the object when it is deformed (Cootes, T. F. et al. 1992).

To generate the point distribution model, key points of the object are identified manually in the training set. There can be as many points as required and they do not have to be on the boundary of the object. The modes of variation between the points are calculated and allow the model to be deformed in order to obtain a match. The points that can be deformed are given a range of variance for the computer to cycle through (Cootes, T. F. et al. 1992). Each point is assigned a type (Cootes, T. F.Taylor et al. 1994). Type 1 points are application-dependent points representing significant features of the shape. Type 2 points represent application-independent features that may appear in certain orientations and type 3 points are interpreted from type one and type two points. Their position is calculated based on the position of associated type one and type two points and by incorporating a degree of variation learned from the training set.

To improve the reliability and accuracy of active shape models, the use of statistical grey-level models may be incorporated into the ASM process. When defining a point, the assumption can be made that the grey levels around the point are going to be similar across various images (Cootes, T. & Taylor 1993). The grey-levels around the point can be modelled with a mean degree of variance and a number of modes of variation (Cootes, T. & Taylor 1993). To further speed up the process, Cootes et al (1993) describes how a Bayesian classifier can be applied to identify the area of the image most likely to contain the point.

Incorporating statistical grey-level searching into the ASM technique, increases the reliability and accuracy of the technique. The use of a sub image, albeit a grey-level image, to help in the point identification process, means that the resolutions of the training images are going to have an impact on its effectiveness and speed. The use of a multi-resolution approach can also help to speed up the process and make it more accurate. Essentially, the search should start by making large jumps in an attempt to locate the general area of the image that the sub-image is likely to appear in. Then the resolution can be increased and the search refined until a match is found (Cootes, T. F., Taylor & Lanitis 1994).

2.6 ASM Toolkit

The ASM toolkit is a set of Active Shape Model tools developed by Cootes. It is a standalone application and includes features such as the ability to input a point distribution model manually and a range of grey-level models to help identify points in new images. Cootes developed the ASM toolkit for research purposes which is freely available to download.

Kuksin (2001) analysed the ASM toolkit and how appropriate it was for use in identifying starfish in underwater images. Initially the ASM toolkit was run on the images with no pre-processing. This showed that the toolkit was unsuitable for the images in that form. Kuksin then pre-processed the images, enhancing the contrast and identifying the regions that may be starfish. However, there were still no clear results. Kuksin attributes this to the poor quality of the images and the lack of contrast. To further improve the results, higher quality images with suitable pre-processing should be used. Kuksin also indicates that having more points on the ASM model may help.

2.7 ASM Toolkit Underlying Operation

The ASM Toolkit is used to create, train, build and view active shape models and active appearance models. This is the core of the project and has formed the main part of the research that has been carried out.

The ASM Toolkit takes a set of training examples and analyses the saved points files. It transforms the shapes to minimise the difference between them in terms of size, orientation and location. This minimises the sum of the squared distances to the mean of the set. Once this analysis has been performed, each shape can then be represented as a vector with 2n elements containing the coordinates of all the points in the shape.

The vectors representing this training set form a cloud in 2n dimensional space and can be considered a sample from a probability density function. This cloud represents all known variances in the shape of the object and can be analysed with Principal Components Analysis to determine the major axes of the cloud. This creates a model of the most common variations between points.

The shape model is then in the form of a point distribution model containing the mean shape of the training-set, unit vectors along the major axes of the cloud, and a vector of shape parameters that describe possible deformations.

In order to find new instances of the object using the model, the shape parameters can be varied within the limits learned from the training-set to generate new examples. These can then be matched in the local area around the model to find new instances.

2.8 Summary

With the current growth in marine science, automation of tasks is becoming more and more important. There is simply too much data being collected to analyse manually. Some of the applications that could potentially be automated include identifying objects in images, autonomous control of underwater vehicles observing environments and animals, and filtering out images with no interesting objects or events. This is why image processing and computer vision are becoming important in many marine applications.

There are several techniques being researched to identify objects in images. The active shape model approach can theoretically be used in any appropriate application. It requires a certain amount of initial training for the computer to learn the possible variations in shape of the object it is looking for. Once this has been done, it should be able to identify the object in any orientation.

While the ASM technique was developed for face recognition and medical imaging applications, it should be possible to adapt it for use with underwater images. Once it has learned a shape model for the object it is searching for, with the possible shape variations, it should be able to cope with swimming fish. The contrast and level of detail in the underwater images may be a problem, but so long as the images are of a high quality and appropriate pre processing is carried out, it should be successful.

Chapter 3

Methods

This chapter will outline the aims and requirements of the project and its individual components. The input and output from the active shape model toolkit will be discussed as well as the wrapper application that manipulates the data for the toolkit. The final section will discuss the evaluation techniques that were used.

3.1 Initial Considerations

3.1.1 Aim and Requirements

Now that video data collection is increasing in importance in the marine science circle, a vast amount of data is being collected but it cannot be processed fast enough with current manual techniques. The main aim is to be able to identify a fish in an underwater image using only computer vision techniques. This involves firstly identifying the region of the image that may contain a fish. Once this has been done, a local search using the active shape model (Cootes et al. 1994) technique can be performed to find the fish. The overall aim is to produce meaningful data about the identified fish, such as its length.

3.1.2 Adaptable Shape Model Approaches

There are several approaches to utilising the adaptable shape model technique. Initially it was thought that a simple ellipse might be enough to match a fish due to its shape. However, this approach would not have been very accurate or able to handle occlusions effectively.

The second approach was to implement the active shape model (ASM) (Cootes et al. 1994) technique using a tool such as MATLAB. This was the preferred method until the ASM toolkit was found which had the majority of the functionality already implemented. This would save a lot of time and allow the process to be further automated.

3.1.3 Development Platform

In order to write the wrapper application, some considerations were made regarding the platform and development time. The preferred option was to write the application in MATLAB as it already had many of the rotation and scaling methods implemented. There were two issues at the time regarding the use of MATLAB. The first was that the licence was only for the Macintosh operating system and the ASM Toolkit had only been developed for the PC. Secondly, the learning curve involved as compared to an already know language was just too large. Based on these two factors, Java was chosen because of its cross platform nature and wide acceptance in industry.

3.2 ASM Training

Before the model could be trained, a set of training images was carefully chosen from the images provided by AQ1 Systems. These images were selected so that the model could have the variations in shape that a fish may present to the camera as it

swam. Forty images were selected and divided into two categories. The first category contained 20 well-defined, well-contrasted typical samples that could be easily identified and traced. The second set of 20 samples was of less quality and provided a greater spread of samples.

In order to build a model from the training-set, an ideal image needed to be traced to initially locate the key points of the fish (Figure 3.1).

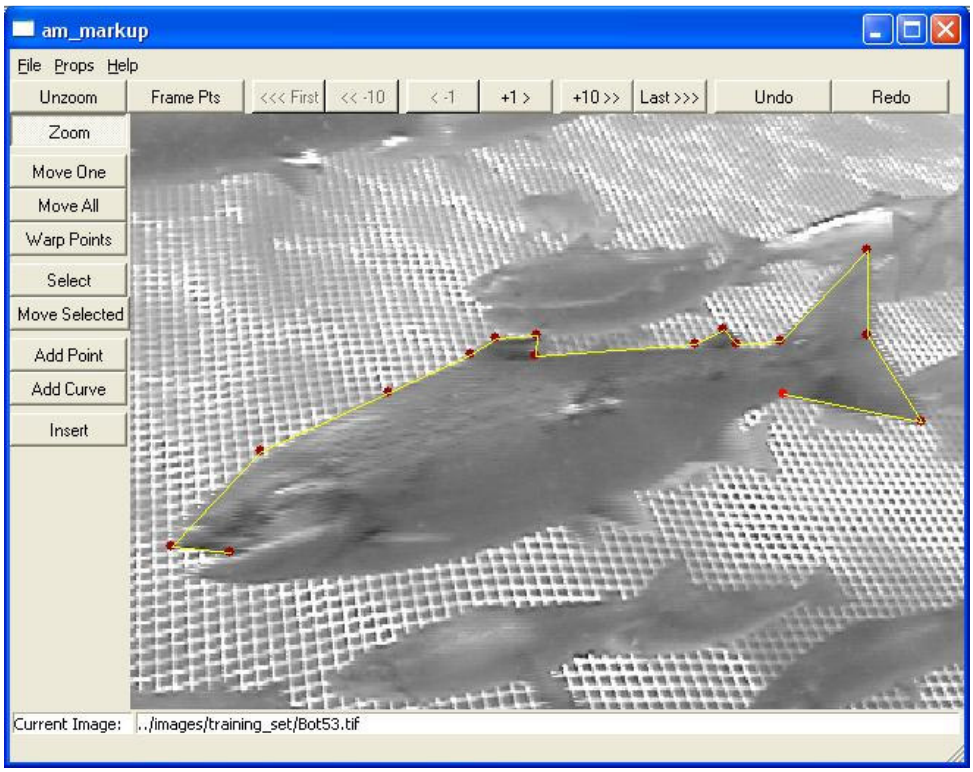

Figure 3.1 Defining the point locations from the first sample**.**

(Image loaded in am_markup interface is the property of AQ1 Systems.)

Each defined point identifies the same feature of the fish in every instance. Using the "am_markup" tool from the ASM Toolkit, each image in the training-set is loaded and the points are moved into position manually for each fish sample (Figure 3.2). Once the points around a fish have been set, a points file is generated for that instance saving the coordinates of each point (Figure 3.4).

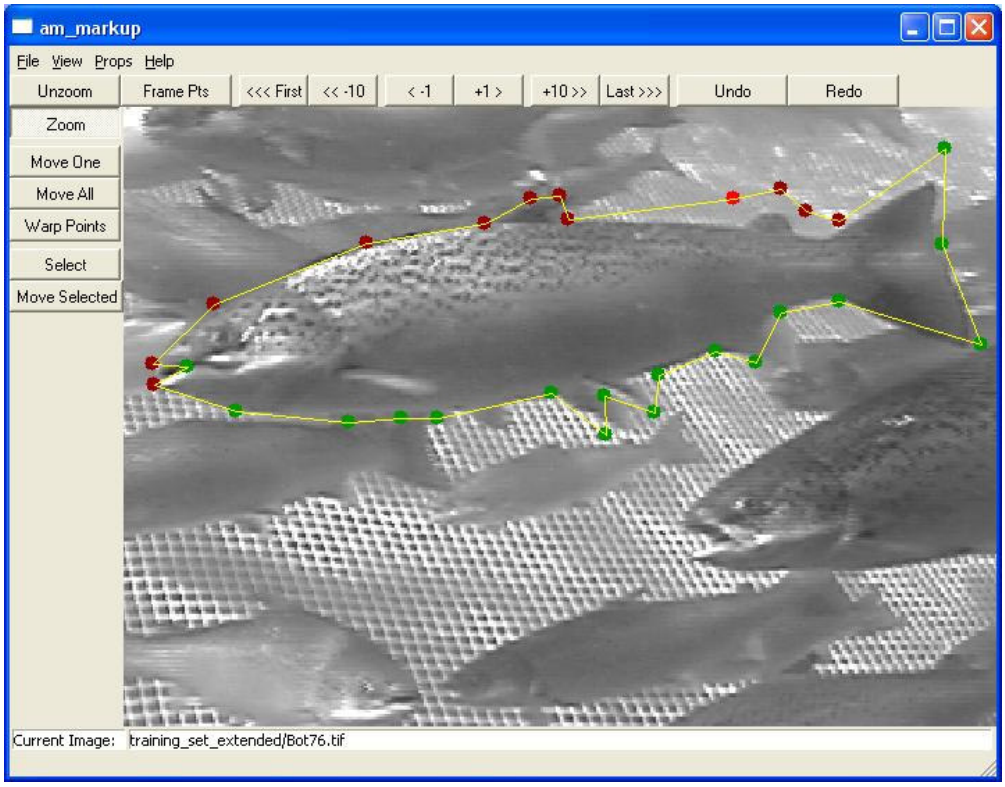

Figure 3.2 Transforming the points around a sample in the training set.

(Image loaded in am_markup interface is the property of AQ1 Systems.)

Once all the training is complete, the model can be built using two programs included in the toolkit. Firstly the "am_build_apm" tool from the ASM Toolkit is run. This reads the data from the individual points files (Figure 3.4) and builds an appearance model from the points and images in the training-set. There are three components that define the appearance model when it is built. The first is a statistical shape model, which is a multi-resolution model defining the modes of variation between the points. The second component is a statistical texture model defining how the textures are sampled and built. The third component is a combined model defining the relationship between the shape model and the texture model. To complete the active appearance model the "am_build_aam" tool from the ASM Toolkit is run. This builds an active appearance model using the appearance model from the first step.

3.3 Implementation

This section will discuss the process from input to output including the process of searching for a fish within an image. This will include details about the ASM wrapper and how it works.

3.3.1 ASM Wrapper Program

The ASM wrapper (Appendix D) is a program written specifically as part of this research to streamline the searching process for the user. It takes input from a file listing various images and details about a region possibly containing a fish. This data is formatted and saved to another file for the ASM Toolkit to load and perform a search on. The process is described in the following sections.

Wrapper Input

The file that the wrapper loads containing the list of images and region information is in the format (image filename|centre x|centre y|horizontal length|orientation). This can contain any number of lines describing these possible regions. An example is shown below (Figure 3.3).

```
bot 1.tif|166.229599|190.065007|90.957374|0.090365|
bot 2.tif|372.518251|403.515998|120.944325|0.429161|
bot 6.tif|238.956989|119.180492|91.608746|0.084957|
```

Figure 3.3 Input format for the wrapper program**.**

Each line in this file describes a segment that was identified as a possible fish sample. The first component is the name of the image in which the sample was identified. The second and third components are the x and y coordinates of the

centroid of that sample. The fourth component is the length of the sample in pixels and the fifth component is the rotation in radians.

Point Transformation

The wrapper program loads a default points file similar to Figure 3.4 with 29 points identified. The points in this file describe a typical sample, horizontally aligned, on which the transforms can be based.

Once the default points file has been loaded, the wrapper carries out each operation to position the model as best it can, based on the input in Figure 3.3. The translation operation offsets all the point locations so the centre of the model matches the centre of the identified region. It then rotates the points around the centroid based on the orientation component of the input. Finally it scales the points from the centroid, based on the horizontal length component of the input.

Scaling of the model does have one limitation that affects the final scaling step. The model can only be linearly scaled within the ASM Toolkit not allowing for fine-grained adjustment. As a result of this, the scaling needed to be based on either the length of the sample or the height. The length was chosen as it was greater than the height in all samples and would therefore be more accurate for the model to scale itself around. When the model was scaled to fit the length of the sample, the height was proportionately scaled as well allowing the model to roughly fit the sample in both dimensions.

Wrapper Output

Two data files were manipulated as the program ran. The first was the individual points file generated by the wrapper program. This was used to position the fish model over a sample in the ASM Toolkit. The format of this file is shown in Figure 3.4.

```
version: 1
n_points: 5
{
52.6582 240.759
25.3165 235.443
120.253 166.329
168.101 147.342
184.051 136.709
}
```

Figure 3.4 Example of a points file containing 5 point locations.

The numbers on each line define the x and y coordinates of each point in the model. This example contains 5 points, but the actual model (Appendix A) contained 29 points to define the fish shape.

The other file manipulated by the wrapper was the Shape Model Data (SMD) file containing a list of points files together with the corresponding image. This was the most relevant information in this file. However, it also contained other information such as program directories and model parameters. An example of the main part of this file is shown in Figure 3.5, excluding the irrelevant parts. The full SMD file used in the system is listed in Appendix B. However these were not the files produced by the wrapper as the wrapper generated them automatically.

```
// Details of points : images
training_set:
{
  Bot53.pts : training_set/Bot53.tif
  Bot22.pts : training_set/Bot22.jpg
  Bot22_1.pts : training_set/Bot22.jpg
}
```

Figure 3.5 Example of a Shape Model Data (SMD) file containing 3 entries.

3.3.2 Searching Process

The searching process was carried out using the "am_markup" tool from the ASM Toolkit. The model was first loaded by specifying the model name at the command line. Once the program was started, the SMD file was manually loaded containing a list of the testing-set of images and points files to be searched.

Manual Positioning of the Model

The SMD file that was loaded contained a list of images with a blank points file next to each one. This meant that when each sample was loaded, the model needed to be manually positioned over a fish. This was done using the left and right mouse buttons while dragging. Once the model lay approximately over the fish (Figure 3.6), the search could then be initiated with the "Search" button. The model attempted to fit itself to a fish in the local area.

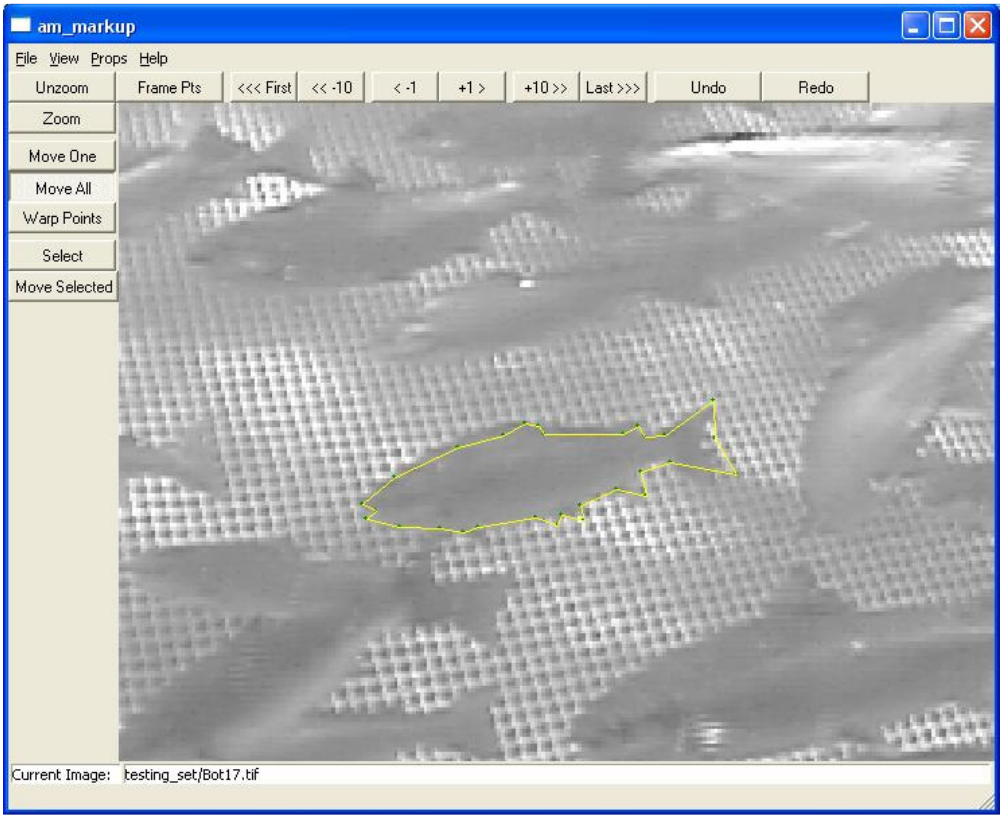

Figure 3.6 Example of the model being manually positioned.

Manually positioning the model could have lead to potential consistency problems between the tests by not being able to position the model in the same starting position for each test run. To avoid this, the location of the model for each sample was preset by manually transforming the model over each sample and saving its current points to a points file. This also generated an SMD file containing references to the corresponding points files for each sample. This allowed the starting location for each sample to be the same and that the same set of samples would be identified in every test run. This also meant that once the model had been manually transformed over each sample in the testing-set and their points saved, the process was exactly the same as the automatic method below.

Automatic Positioning using Wrapper Output

This followed a similar process, except that the SMD file also contained references to points files generated automatically by the wrapper program. The points in these files correspond to a region in the image identified as a possible fish. When each image was loaded into the ASM Toolkit, it also loaded this points file, displaying a set of blue dots over the identified sample (Figure 3.7). The “Fit to Labels” button was pressed to align the model with these blue points. Following this, the “Search” button was pressed to initiate the search process over the local region of the image.

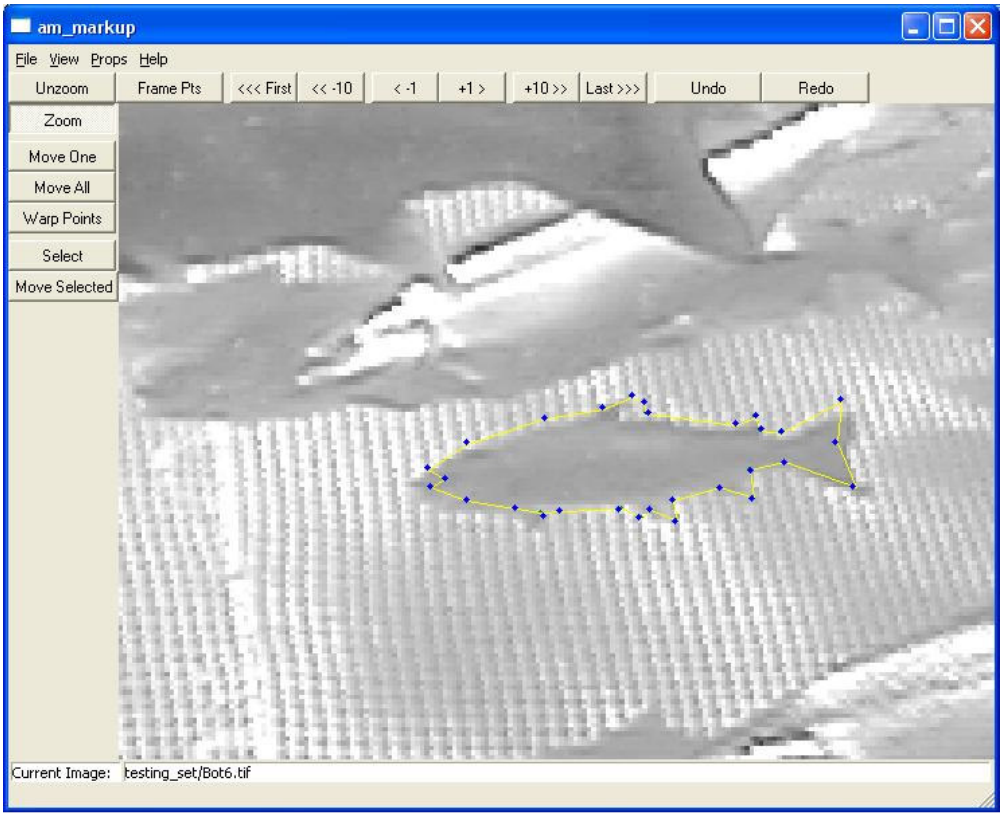

Figure 3.7 Example of the model being automatically positioned.

(Image loaded in am_markup interface is the property of AQ1 Systems.)

Further Output From the ASM Toolkit

When the model matched a fish (Figure 3.8), the new points identified by the search could then be saved to a new points file for that image by choosing "Save Points" from the file menu. This file contained the points of the model and should have very closely outlined the identified fish. This could then be loaded as input to another program so appropriate calculations, such as the length of the fish, could be carried out.

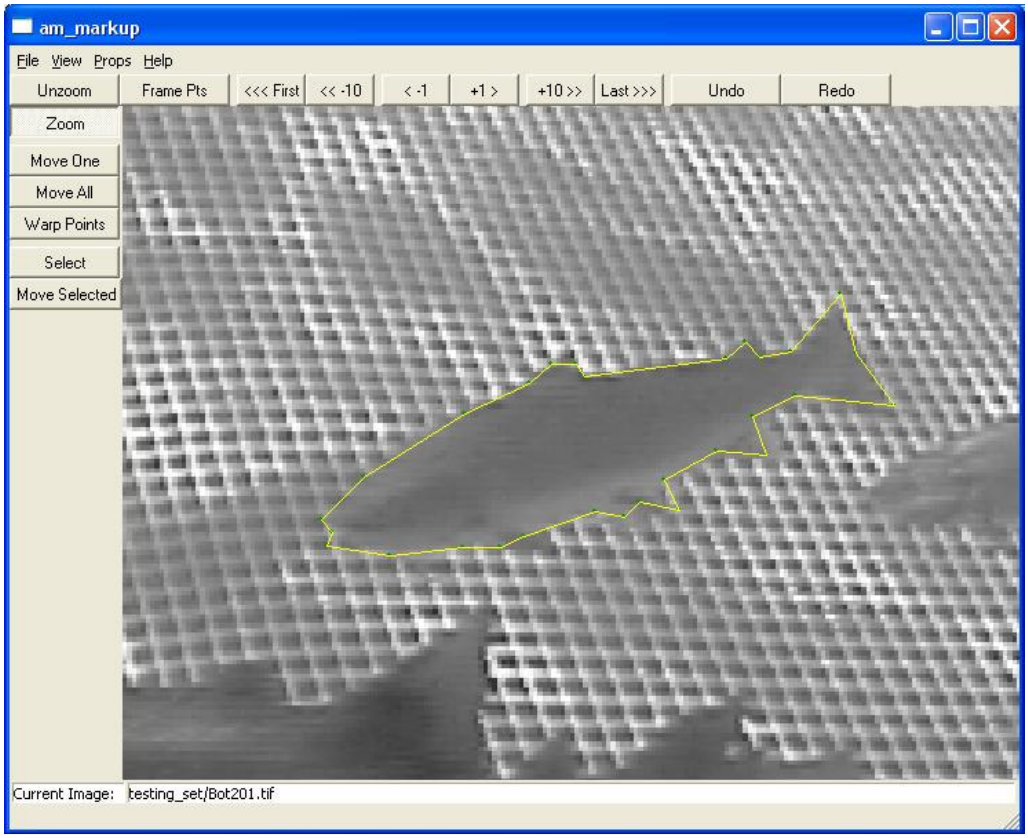

Figure 3.8 Example of the model perfectly matching a fish sample.

(Image loaded in am_markup interface is the property of AQ1 Systems.)

3.4 Evaluation Techniques

Before any evaluation could begin, the testing-set of images had to be defined. This was done by first taking out all of the images used in training the model. Then the rest of the images were put into good, medium, and poor sets based on several criteria used to grade how good they were. The criteria were based on the level of contrast in the image, how well defined the fish were, and how many fully non-occluded fish appeared. Each image was subjectively compared against these criteria and assigned to a category. From these, a random set of images was chosen with 20 images coming from each set. This new combined set was labelled the testing-set and contained a total of 60 images. This set was used for every test carried out and was also used by Lambert (2005) for consistency.

The first test performed was on fully non-occluded fish. These were defined as fish that had no other fish in front or behind them. The only thing in the background around the fish was the fish net of the ring. An example is shown in Figure 3.9 where the fish in the centre of the ellipse is fully non-occluded. The search was then carried out and the results collected. This gave an indication of the best possible performance of the model.

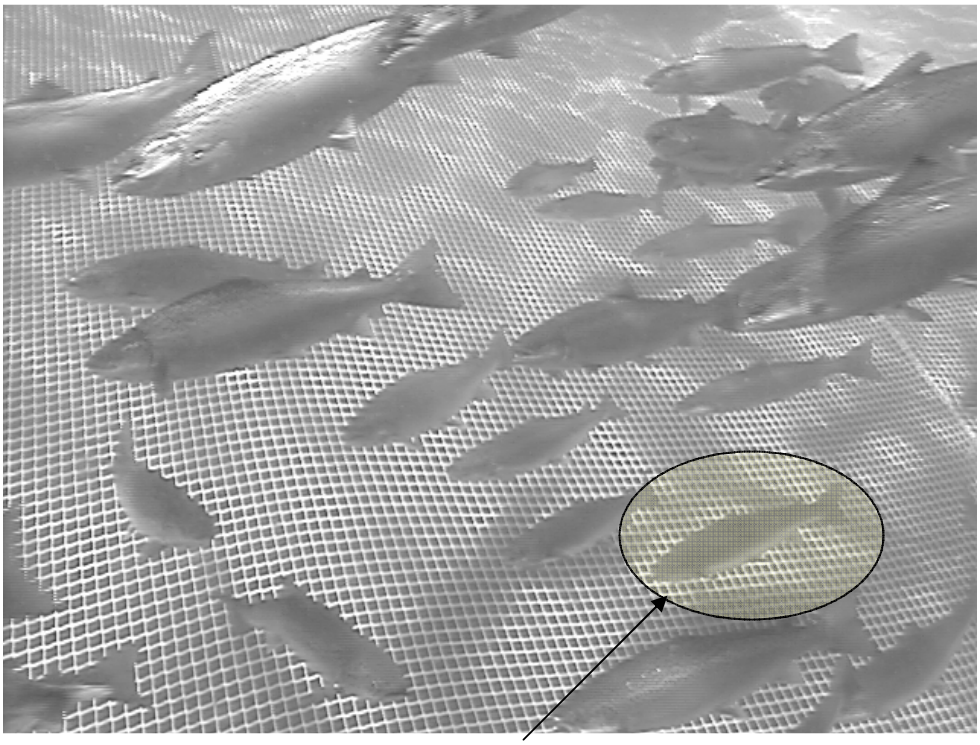

Figure 3.9 Example of fully non-occluded fish sample.

(Image is the property of AQ1 Systems.)

The second test was performed on almost non-occluded fish. These were defined as fish that had at least one fish in the background overlapping. An example of this is shown in Figure 3.10 where the sample inside the ellipse has another fish behind it. This gave an indication of the performance of the model in a more common scenario.

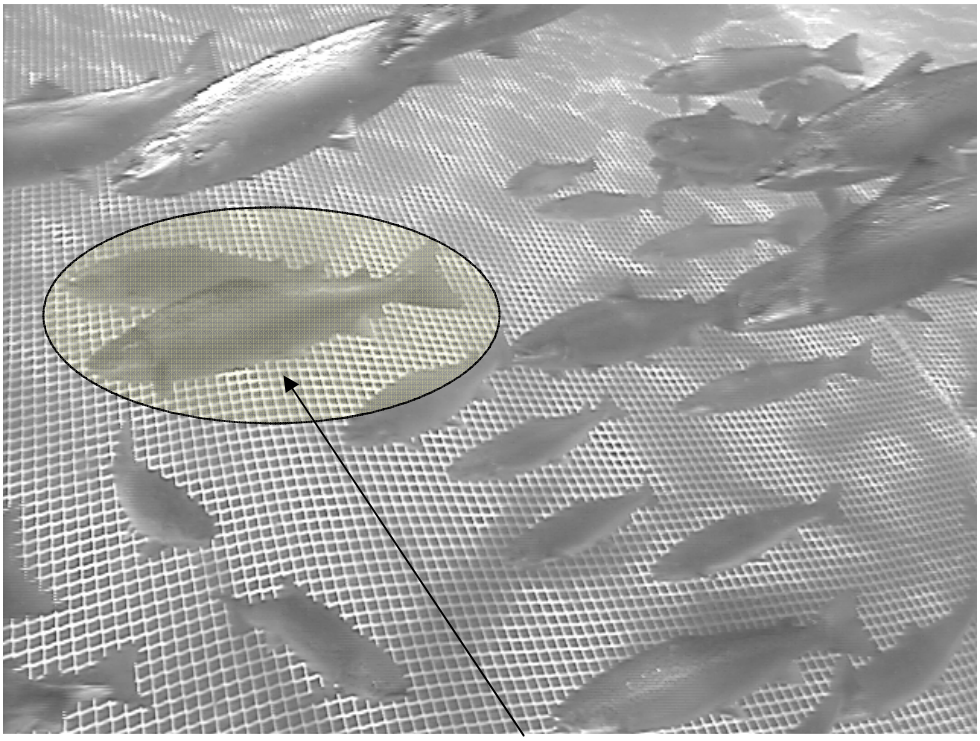

Figure 3.10 Example of almost non-occluded fish sample.

(Image is the property of AQ1 Systems.)

The last set of tests involved taking the output data from the wrapper application and performing the search on each instance listed in the SMD file. The regions identified as possible fish did not always contain an appropriate sample. In some cases it located two fish that were overlapping or a fish swimming in the wrong direction. These were still included in the results to give an indication of the real world performance of the system at this stage.

The result of each search attempt was assessed by eye using consistent evaluation criteria. Generally it was clear if a match occurred in each case. However, there were also some borderline cases. Figure 3.11 shows an example where the model does not quite fit the sample exactly, but is close enough to classify as a match. Figure 3.12 shows an example of a non-match where the model did not fit the sample well enough.

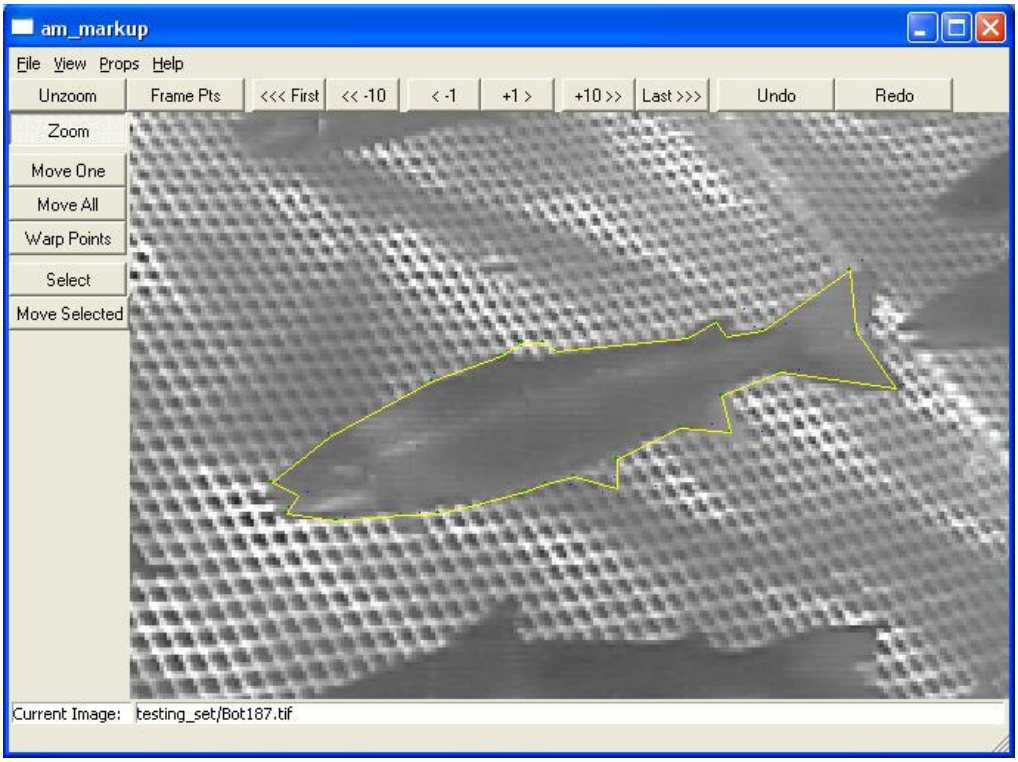

Figure 3.11 Example of a borderline case that was classified as a match.

(Image loaded in am_markup interface is the property of AQ1 Systems.)

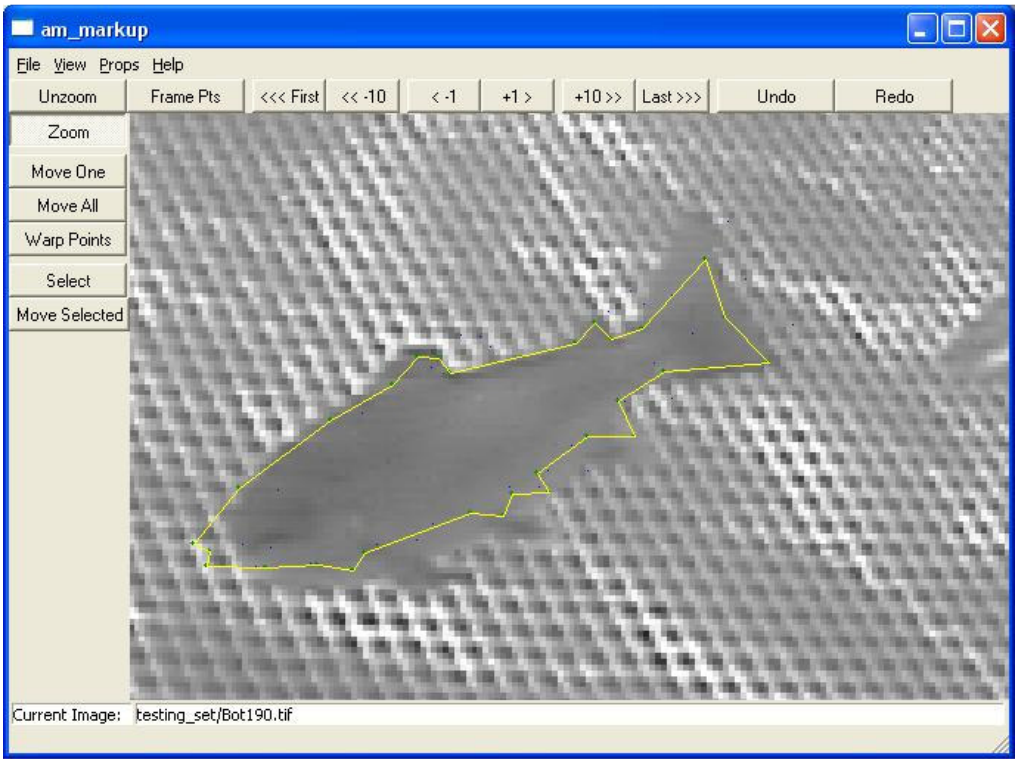

Figure 3.12 Example of a borderline case that was classified as not matching.

(Image loaded in am_markup interface is the property of AQ1 Systems.)

Two evaluation criteria were used in deciding whether the model matched a sample or not. The first was how closely the model matched the features of the sample such as the top and bottom fins and the body. In some cases the minor fins above and below did not match, but this was not considered to be important as long as the body was a close match. The second more important criterion was how well the model matched the length of the sample. If the model did not accurately match the nose of the fish or the tail, it was considered as a non-match. This was because the length of the sample will ultimately be important to the system and needs to be accurate. The borderline case in Figure 3.11 managed to match the top of the nose well enough, and was barely close enough with the tail. But the case in Figure 3.12 did not match either the nose or the tail close enough to classify as a match.

Chapter 4

Results and Discussion

Results were obtained from several different categories within the testing-set of images. Depending on how well the model matched the fish in the image, a score was given to that instance of either 0 or 1. If there was a partial-match, it was given a score of 0. Only the complete matches were given a score of 1 (Appendix C).

4.1 Data Set Construction

4.1.1 Data Set: Category A

Category A consisted of samples where there were no fish in front of or behind them. The only thing directly behind a fish was the fish net. This category included ideal samples that were fully non-occluded and are representative of the best possible set of samples likely to be presented to the system in the real world.

This set was manually identified from the 60 testing images by transforming the model, using the mouse input with the ASM Toolkit interface, roughly over a sample and then saving the points to a file. The points were only of the model without any search being performed. This meant that these points could be loaded in the future and the model would be sitting in the same place in every test run for this sample.

4.1.2 Data Set: Category B

Category B included almost non-occluded samples where there was at least one fish behind and no fish in front. This set was manually identified in the same way as category A. This will demonstrate how well the model can handle a more varying background behind the sample and allow a comparison between this and the most ideal set which is category A. This type of sample was much more common in the images than the fully non-occluded type and is the second best type that can be presented to the model. This will significantly extend the number of testing samples allowing a much larger set to be formed.

4.1.3 Data Set: Category C

Category C formed part of an automated real world test. This comprised samples previously found by a segmentation program (Lambert 2005) and with points generated automatically by the wrapper program. This simulated a real world system based on the active appearance model with automated input from another application.

The set comprised only segments that were classified as fully non-occluded according to the segmentation algorithm (Lambert 2005). However, this did not necessarily mean that the samples were technically fully non-occluded due to the algorithm limitations.

The segments identified by the segmentation program were loaded into the wrapper application. This program then generated the points files automatically which were to be loaded by the ASM Toolkit for testing.

4.1.4 Data Set: Category D

Category D formed the other part of the automated real world test. This was comprised of segments that were identified with a further segmentation technique (Lambert 2005). This extended the number of samples found in category C, allowing more samples to be searched as part of the automated test.

4.2 Results – Initial training model

The first test run was based on a model trained with 20 fish examples from the training-set. This mainly included ideal fully non-occluded samples typical of category A. A description of each result is included in the following section.

	Category			
	A	*B*	*C*	*D*
Number of Samples	125	209	95	35
Number of Matches	85	88	54	11
Percent Matching	**68.00%**	**42.11%**	**56.84%**	**31.43%**

Table 4.1 Performance of the initial model in the four tests.

In category A, 125 samples of fish were identified from the testing-set. From these, the model accurately matched 85. This means that 68% of the samples were matched in the ideal (Table 4.1).

Category B comprised 209 samples from the testing-set. The model matched only 88 of them, giving a 42% match rate (Table 4.1).

Category C had 95 samples identified as possible fish from the testing-set by the segmentation program. With this category, 54 samples were matched giving a 57% match rate (Table 4.1).

The category D testing-set contained a total of 35 samples identified by the further segmentation program where only 11 were successfully matched by the model. This gave a match rate of just 31%, which is predictable given the higher percentage of unsuitable samples in this set (Table 4.1).

4.2.1 Discussion

The results from category A show that when the best possible samples are selected from a random testing-set, 68% will match. However, when the training set was extended, this rate went down to 61%, which showed that the training of the model is very important. The initial training was done on 20 ideal fish samples with mainly fish net in the background immediately around the fish. This is why the initial model performed so well in category A where the samples only had fish net in the background.

4.3 Results – Extended training

In order to test the effect of a larger training-set on the model, a sample of 40 additional images were selected to extend the model. These images comprised lower quality samples with poor lighting that had fish in the immediate background, the same as the category B set. This was done to give the model a broader knowledge about the potential sample and test the effectiveness of this approach.

	Category			
	A	*B*	*C*	*D*
Number of Samples	125	209	95	35
Number of Matches	77	94	29	5
Percent Matching	**61.60%**	**44.98%**	**30.53%**	**14.29%**

Table 4.2 Performance of the extended model in the four tests.

The results from category A match 77 samples of fish from the set of 125. This equates to a 62% match rate (Table 4.2).

In category B, 94 samples of fish were matched out of 209, which is a 45% match rate (Table 4.2).

The test on the category C set containing a total of 95 samples, only identified 29 matches. This gave a match rate of just 31% (Table 4.2).

The final test for this model on category D contained 35 samples of which the model matched just 5, giving the lowest match rate of 14% (Table 4.2).

4.3.1 Observations

The 20 extra samples that were selected for the extended training had mainly fish in the immediate background. When this extended model was used on the category A set, it performed worse because the training set comprised more than just the ideal samples. The same reasoning can be applied to the category B set but in reverse. Since the testing samples had more than fish net in the background, the initial model performed poorly as it was not trained on similar samples. The extended model was trained on better samples that included fish in the background and therefore, performed better on the test set.

4.4 Combining The Manual Input

The category A set and the category B set do not contain any common samples when their sets are combined. This allows the data to be compared to show overall trends from a large set of samples.

The results in category A show a 6% drop in performance when the model is trained on twice as many samples. The category B set shows a 3% gain with the extended model. Category A and category B can be combined for each model giving 334 samples. For the initial model trained on 20 samples, there were 173 matches giving a 51.8% match rate. With the extended model trained on 40 samples, there were 171 matches, which gave a 51.2% match rate. The individual results and the combined results are displayed below in Table 4.3.

	Category		
	A	B	Combined
Initial Training	68%	42%	51.80%
Extended Training	62%	45%	51.20%
Difference	-6%	3%	-0.60%

Table 4.3 Match rates when the manual test results are combined.

These two models vary by just 0.6% when the results are combined, which is insignificant given the size of the sample set. These differences are graphed in Figure 4.1 showing the closeness of the results when the categories are combined.

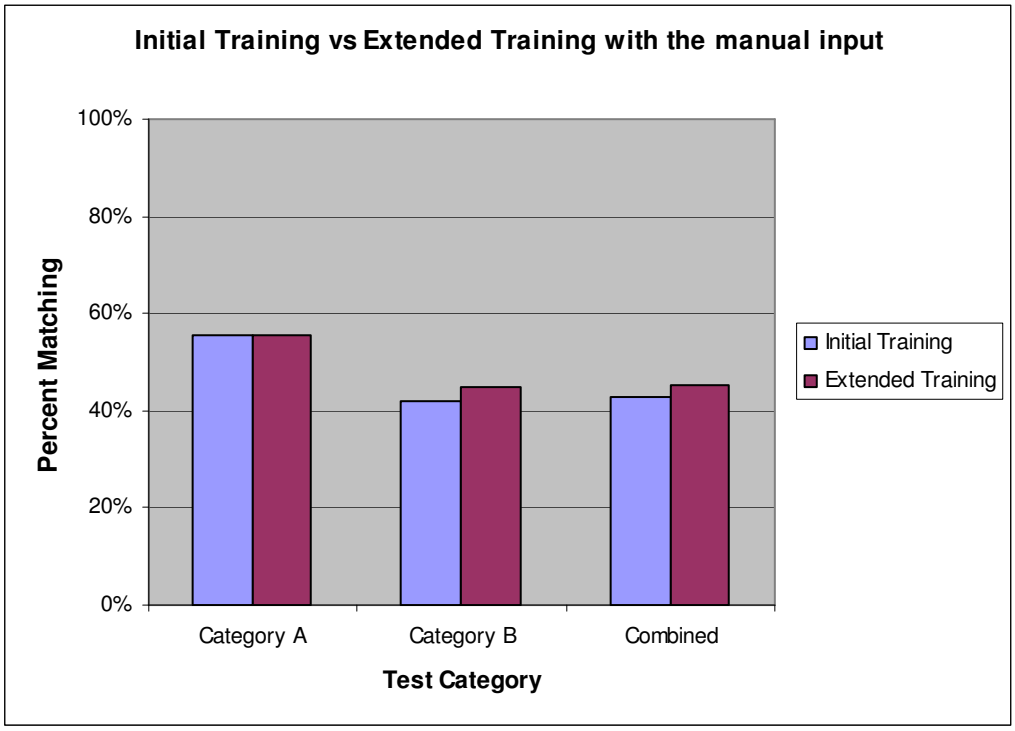

Figure 4.1 Comparison of the two training models in the two manual tests.

4.4.1 Observations

The results from this test give an indication of the difference between the models. The initial model performed better on the category A set because that was the type of image used to train the model. However, with the extended model trained on 20 extra samples that were typical of category B, it performed better on that testing set. This shows that the number of training images plays a smaller role on the results than the actual spread of the images. Training only needs to be done on enough samples for the model to know the main deformations a fish may present to the camera as it swims.

4.4.2 Discussion

When the results are combined from the two categories in each test run, the difference between the initial model and the extended model is negligible, being 0.6% out of a total of 334 samples. This shows that a model trained for a wide range of lighting conditions, backgrounds and fish samples does not perform very well. However, if the model is tailored for a more specific set of conditions, then it performs much better in those conditions. This is why the combined results are more even as the extended model has a broader knowledge but is less accurate in specific cases. It can match many samples over the two categories, but not a significant number in the individual categories.

4.5 Combining The Automated Input

Categories A and B that are generated automatically can also be combined to give an indication of the performance when these two segmentation techniques are used in conjunction with each other to form one large set of samples.

The results in these categories are much more telling. Table 4.4 shows that in the category C test set, the match rate dropped by 26% and the category D set dropped by 17% when the model was trained on an additional 20 images. When these two categories were combined for each training model, 130 samples were searched. With the initial model trained on 20 samples, there was a combined total of 65 matches giving a match rate of 50%. With the combined results of the extended model trained on 40 samples, only 34 samples were identified giving an overall match rate of just 26%. This results in a 24% drop in the number of matches when the results are combined for the extended model.

	Category		
	A	*B*	*Combined*
Initial Training	57%	31%	50%
Extended Training	31%	14%	26%
Difference	**26%**	**17%**	**24%**

Table 4.4 Match rates when the automated test results are combined.

The graph in Figure 4.2 shows the difference in percentages between the two models in the automated tests.

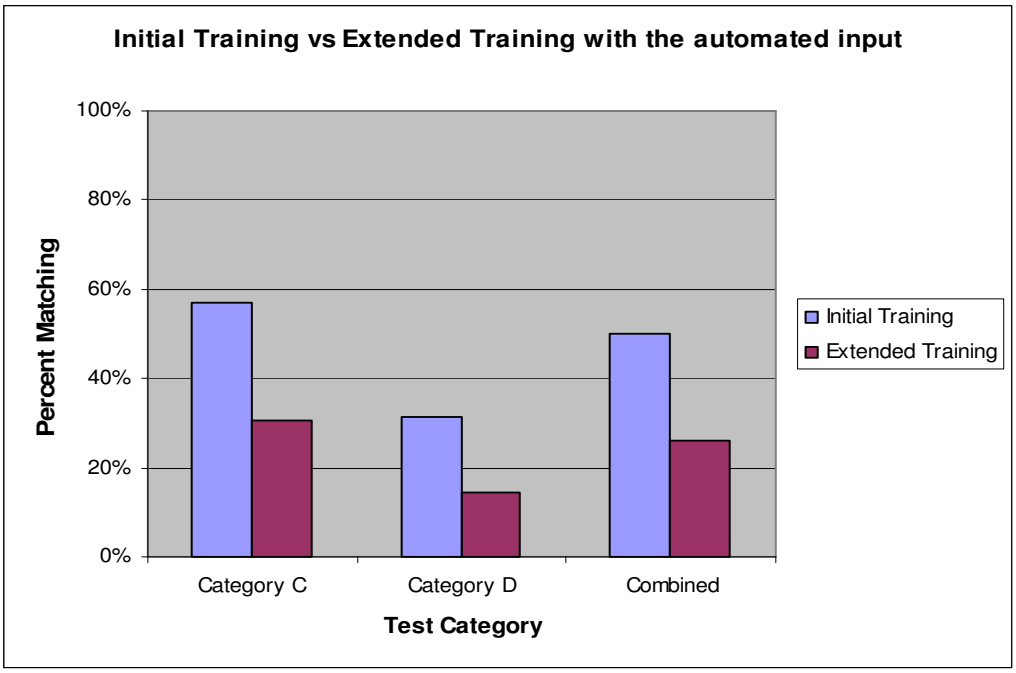

Figure 4.2 Comparison of the two training models in the automated test.

4.5.1 Observations

The results from the automated categories were always going to be poorer due to samples that were identified as possible fish but were not appropriate for the model to match. These could have been fish swimming in the wrong direction, or two fish close together that were identified as one. The other factor affecting these results is that a majority of the samples were fully non-occluded, which the initial model has previously shown to be much more successful in category A. Category C contains a much greater number of samples than category D and as a result, has more effect on the combined result.

The combined match rate with the initial model is 50% in the automated tests, and 51.8% in the manual tests. If the impossible samples could be removed from the automated set (as they do not exist in the manual sets), then the automated set would be even closer to the manual sets, if not better.

4.5.2 Discussion

The result from Lambert (2005) gives an idea of how the model will work in a real world application. The ASM Toolkit loaded input from the wrapper program describing a region that possibly contained a fish, and tried to match the model over that area. The results from category C show that the model can match 57% of regions that were identified as fish. In general, the samples in category C were fully non-occluded and can therefore be compared to category A that had a match rate of 68%. Given that the category C set contained a number of unsuitable samples that would not have been included in category A, when these are disregarded the difference between the two is less significant.

In terms of the category D set, the results from this tend to be low. However, given the small size of the set, it does not greatly affect the overall performance. The reason for it being so low is that the set contains a much higher number of unsuitable samples that the model simply cannot match.

4.6 Comparison Of The Models

The graph in Figure 4.3 outlines the performance of each model over the four test categories. In the two manual sets (categories A and B), the models perform closely overall, although there is a difference in the individual tests that is eliminated when they are combined. This graph also includes an extra model that was only tested on category A. This was trained on the same samples as the initial model, but with five omitted samples to test the effect of a 15-sample model.

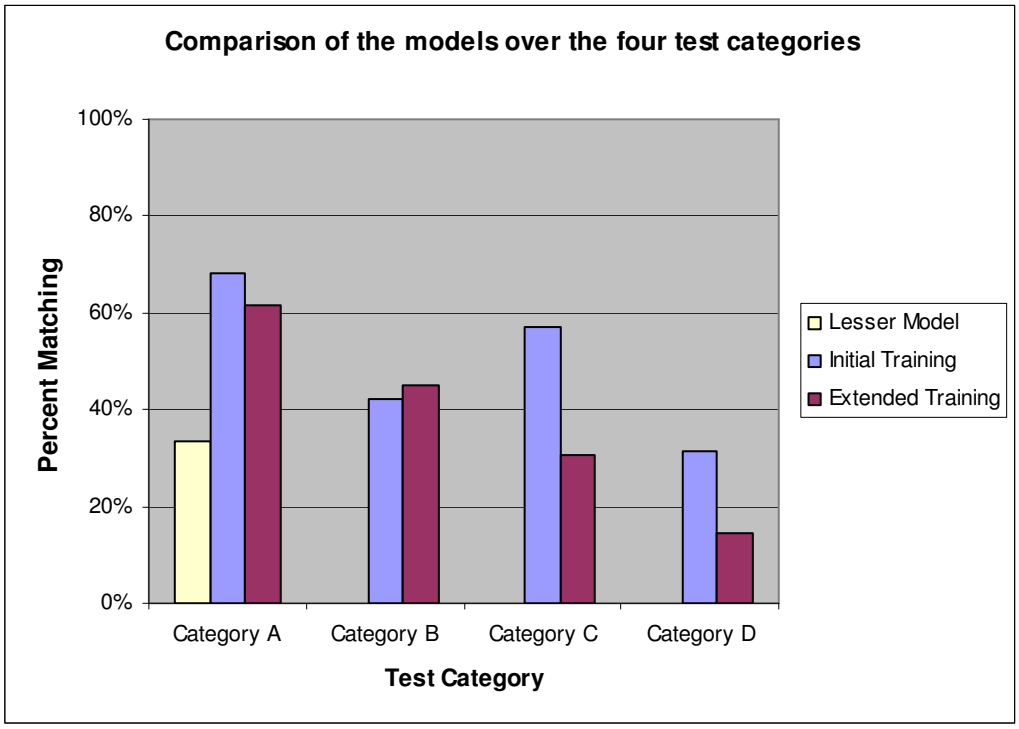

Figure 4.3 Comparison of the models in each test set.

4.6.1 Discussion

The result here from the lesser model is very significant. By training the model on 15 samples instead of 20, it lowers the performance by over 20%. This indicates that the size of the training set for fish should be around 20. By using 20 samples, the model

will gather enough knowledge about the fish for successful identification in the majority of cases.

This graph also shows that the initial model outperforms the extended model quite significantly in three of the four tests. The most notable are the two automated tests where the initial model almost doubles the performance of the extended model.

The other result that can be mentioned is the difference between category A and category C. These two testing-sets were similar as the samples in category C were generally fully non-occluded. This allows a comparison to be made between the two. With the initial model, the two tests score highly, although the category C test is slightly lower.

4.7 Total Fish In Testing Set

The samples that were identified in each test category all came from the 60 testing images. These samples were only a small proportion of the total number of fish depicted in the images. Some of the images contained so many fish that they were too blurred to distinguish and include in the count. The figures below in Table 4.5 give an indication of the distribution of fish between the images.

Number of images:	60
Total fish in testing images:	1122
Minimum number in an image:	8
Maximum number in an image:	46
Median:	16
Standard Dev:	7.4

Table 4.5 Rough count of the total number of fish in the testing set of images.

In the 60 images, 1122 fish were clear enough to distinguish as a fish by eye. The lowest number of fish depicted in a single image was 8 and the highest was 46. However the median was 16 with a standard deviation of 7.4. This indicates that there is a good chance that the images will contain a reasonable number of fully non-

occluded samples such as contained in category A and also typical of those in category C.

In comparison, category A contained a total of 125 samples, which was 11.1% of the total number counted in the testing set. Category B contained 209 samples, which was 18.6% of the total number. If these two numbers are combined, 334 samples were included in these categories. This equates to 29.7% of total fish identified in categories A and B.

Given that the initial model managed to match 68% of the samples in category A and that 11.1% of the total fish in the 60 images were included in that category, it can be calculated that the initial model matched 7.5% of total fish depicted in the 60 testing images. If the same calculation is applied to the combined results for category A and category B, the initial model matched 17.4% of the total fish depicted in the testing images.

4.8 Further Discussion

4.8.1 Pre-processing

When considering these results, it should be remembered that the images were taken straight from the camera. There was no pre-processing done on the images before the model performed its search. The effect of pre-processing is hard to predict. However, there were instances where the model should have matched a sample but was unable to do so which could have been the result of poor contrast and blurred edges.

Chapter 5

Conclusions and Further Work

5.1 Conclusions

When the model was used with category A, it performed very well. It managed to match 68% of the samples, which worked out to be 7.5% of the total number of fish depicted in the testing set. This result alone would be satisfactory if the program could be allowed to run over a reasonable length of time. When category A and category B were combined, this went up to 15.4% of total depicted fish.

Essentially, these numbers are quite low, but given the task being performed and running it over any length of time, the desired result would be obtained eventually. It may take a day to collect enough matched samples to accurately calculate the average size of the fish in the ring, but it would still require significantly less resources than current methods.

In terms of the training set, it is very important to select appropriate samples to train the model. The actual number of samples required is dependent on the object being identified and how many variations it can have. With the fish identification task, approximately 20 samples were required to achieve a satisfactory model.

The importance of the specific samples used is shown in the variation in success rate when using different combinations of samples in the training set. When the model is built with just 5 samples removed, it performs significantly worse.

The training set must also be trained on samples that are from the same category that will be searched. If the category A is being searched, then it should be trained on images typical of that category, and likewise for the other categories. This is important as the model is not only based on the shape but also the texture around each point. The results show that when the model is trained on a specific category, it performs better in that category when searching.

5.2 Further Work

The model training process used in this work should be carried out on different sets of 15 images to test the effect of varying the training set. This will allow a better understanding of what samples the model needs in order for an accurate representation of the object. In preliminary tests, the model did not perform as well when it was trained on 25 samples, but again, that may have had more to do with the samples used.

Pre-processing the images before any training or searching is carried out could potentially increase the performance of the model. The images that were used in this research were not pre-processed. A preliminary test was run on the testing images with histogram equalisation pre-processing applied but with no change in result. Further testing here should include training the model on equalised images and then running the test again on the equalised testing set.

A significant goal of this research is to contribute to the development of an automated system for identifying fish and other marine animals in video image sequences. While this has been achieved to a degree, there is still much to be done. The actual searching process still needs to be carried out manually because the ASM Toolkit only accepts mouse input. In order to further automate the process, there are

two options. First, a new input method could be written into the ASM Toolkit, but that would require access to the source code of the ASM Toolkit. Alternatively, the searching algorithms in the program could be implemented from scratch if sufficient information could be sourced. This would be ideal to allow a greater level of customisation in the future. Either way, this will be a limitation until the input method is customised.

Eventually, the adaptable shape modeling techniques discussed here should be included into a fully automated fish searching application. To this end, further calculations can be carried out on the newly searched points. Once the model has identified a sample, the new points around that sample can be saved into another points file. This file can then be loaded by an application to calculate the length and height of the fish. The extension to this would be to identify the same sample in stereoscopic images and then calculate the actual dimensions of the fish and its weight. The images that were supplied were already stereoscopic in that each frame had two image files, one from each camera. Once the details about the camera placement were known, it would be possible to calculate the actual dimensions of the fish.

Chapter 6

References

Ballard, DH & Brown, CM 1982, *Computer Vision*, Prentice-Hall, New York.

Cootes, T & Taylor, C 1993, 'Active shape model search using local grey-level models: A quantitative evaluation', paper presented to British Machine Vision Conference.

Cootes, TF & Taylor, CJ 2004, *Statistical Models of Appearance for Computer Vision*, Imaging Science and biomedical Engineering, University of Manchester, Manchester.

Cootes, TF, Taylor, CJ & Lanitis, A 1994, 'Active Shape Models: Evaluation of a Multi-Resolution Method for Improving Image Search', paper presented to British Machine Vision Conference.

Cootes, TF, Edwards, GJ & Taylor, CJ 1998, 'Active Appearance Models', paper presented to 5th European Conference on Computer Vision, Freiburg, Germany, 1998.

Cootes, TF, Taylor, CJ, Cooper, DH & Graham, J 1992, 'Training Models of Shape from Sets of Examples', paper presented to Proc. British Machine Vision Conference.

Cootes, TF, Taylor, CJ, Cooper, DH & Graham, J 1994, 'Active Shape Models - Their Training and Application', *Computer Vision and Image Understanding*, vol. 61, no. 1, pp. 38-59.

Cootes, TF, Hill, A, Taylor, CJ & Haslam, J 1994, 'The Use of Active Shape Models For Locating Structures in Medical Images', *Image and Vision Computing*, vol. 12, no. 6, pp. 355-66.

Dirk, W, Edgington, DR & Koch, C 2004, 'Detection and Tracking of Objects in Underwater Video', paper presented to IEEE International Conference on Computer Vision and Pattern Matching (CVPR), Washington, D.C.

Edgington, DR, Salamy, KA, Risi, M, Sherlock, RR, Walther, D & Koch, C 2003, 'Automated Event Detection in Underwater Video', paper presented to MTS/IEEE Oceans, San Diego, CA.

Hinton, GE, Williams, CKI & Revow, MD 1992, 'Adaptive elastic models for hand-printed character recognition', *Advances in Neural Information Processing Systems*.

Kass, M, Witkin, A & Terzopolous, D 1987, 'Snakes: Active Contour Models', *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321-31.

Kuksin, J 2001, 'Detecting starfish in subsea videos using flexible shape models', Heriot-Watt University.

Lambert, T 2005, 'Digital enhancement techniques for underwater video image sequences', Honours thesis, University of Tasmania.

Rife, J & Rock, SM 2001, 'A Pilot-Aid for ROV Based Tracking of Gelatinous Animals in the Midwater', paper presented to Oceans 2001, Honolulu, November 2001.

Tillett, R, McFarlane, N & Lines, J 2000, 'Estimating Dimensions of Free-Swimming Fish using 3D Point Distribution Models', *Computer Vision and Image Understanding*, vol. 79, pp. 123-41.

Umbaugh, SE 2005, *Computer Imaging: Digital Image Analysis and Processing*, 1 edn, CRC Press.

Wilson, A 2003, 'First steps towards autonomous recognition of Monterey bay's most common mid-water organisms: Mining the ROV video database on behalf of the Automated Visual Event Detection system.', Middlebury College.

Appendices

Appendix A Default Points File

```
version: 1
n_points: 29
{
52.6582 240.759
25.3165 235.443
120.253 166.329
168.101 147.342
184.051 136.709
209.114 135.19
207.595 142.785
285.063 139.747
301.772 130.633
307.089 138.987
323.797 138.987
368.608 91.1392
370.886 133.671
395.949 174.684
329.114 163.291
304.81 174.684
301.772 194.43
279.747 195.19
255.443 210.38
259.241 228.608
236.456 224.81
239.494 239.241
216.709 226.329
166.582 243.797
145.316 248.354
121.013 251.392
63.2911 257.468
32.1519 250.633
55.6962 205.063
}
```

Appendix B Shape Model Data (SMD) File

B.1 Initial Training Set

```
// Shape Model Data
model_name: salmon
model_dir: ./
parts_file: salmon
image_dir: ../images/
```

```
points_dir: ../points/
// Alternatives: radial,wolfson,sunras,bmp
// shape_aligner can be None, AlignCoG2D, AlignEuclid2D, ...
shape_aligner: align_similar_2d
shape_variant_maker: -
shape_modes:  { min: 0 max: 30 prop: 0.98 }
tex_modes:  { min: 0 max: 40 prop: 0.99 }
combined_modes:  { min: 0 max: 30 prop: 0.99 }
params_limiter: mdpm_box_limits
{
  sd_limits: 3
}
tex_params_limiter: mdpm_box_limits
{
  sd_limits: 3
}
shape_params_limiter: mdpm_box_limits
{
  sd_limits: 3
}
tex_params_pdf: axis_gaussian_builder
shape_params_pdf: axis_gaussian_builder
app_params_pdf: axis_gaussian_builder
n_pixels: 10000
colour: Grey // Alternatives: Grey,RGB,...
// Texture Sampler can be tri_raw, tri_edge...
tex_sampler: vapm_triangle_sampler<vxl_byte>
// tex_aligner can be None, AlignLinear1D, ...
tex_aligner: align_linear_1d
// shape_wts define how to compute relative scaling of shape & tex.
// shape_wts can be `EqualVar', `EqualEffect',...
shape_wts: EqualVar
// point_finder defines point finder used for ASM search
point_finder: local_models
// tex_model defines type of model to represent texture statistics,
eg: pca, pca+haar1d
tex_model: pca
// Image Pyramid Builder can be gauss_byte, gauss_float, grad_float
...
pyr_builder: gauss_byte
points_pyr_builder: Same
max_im_pyr_levels: 5

// Levels of multi-res model to build :
min_level: 0
max_level: 4

// For ASM profile models :
g_in_len: 3
g_out_len: 3
g_width: 1
sample_int: 1
use_grad: 1
norm_grey: 1
restrict_points_to_parts: 0

default_image_pixel_size: 1
// Details of points : images
training_set:
{
  Bot53.pts : training_set/Bot53.tif

  training_set/Bot25.pts : training_set/Bot25.tif
  training_set/Bot42.pts : training_set/Bot42.tif
  training_set/Bot43.pts : training_set/Bot43.tif
  training_set/Bot46.pts : training_set/Bot46.tif
  training_set/Bot47.pts : training_set/Bot47.tif
  training_set/Bot87.pts : training_set/Bot87.tif
  training_set/Bot88.pts : training_set/Bot88.tif
```

```
  training_set/Bot110.pts : training_set/Bot110.tif
  training_set/Bot111.pts : training_set/Bot111.tif
  training_set/Bot168.pts : training_set/Bot168.tif
  training_set/Bot171.pts : training_set/Bot171.tif
  training_set/Bot174.pts : training_set/Bot174.tif
  training_set/Bot176.pts : training_set/Bot176.tif
  training_set/Bot178.pts : training_set/Bot178.tif
  training_set/Bot179.pts : training_set/Bot179.tif
  training_set/Bot211.pts : training_set/Bot211.tif
  training_set/Bot213.pts : training_set/Bot213.tif
  training_set/Bot229.pts : training_set/Bot229.tif
  training_set/Bot231.pts : training_set/Bot231.tif
  training_set/Top10.pts : training_set/Top10.tif
}
```

B.2 Extended Training Set

```
// Shape Model Data
model_name: salmon
model_dir: ./
parts_file: salmon
image_dir: ../images/
points_dir: ../points/
// Alternatives: radial,wolfson,sunras,bmp
// shape_aligner can be None, AlignCoG2D, AlignEuclid2D, ...
shape_aligner: align_similar_2d
shape_variant_maker: -
shape_modes:  { min: 0 max: 30 prop: 0.98 }
tex_modes:  { min: 0 max: 40 prop: 0.99 }
combined_modes:  { min: 0 max: 30 prop: 0.99 }
params_limiter: mdpm_box_limits
{
  sd_limits: 3
}
tex_params_limiter: mdpm_box_limits
{
  sd_limits: 3
}
shape_params_limiter: mdpm_box_limits
{
  sd_limits: 3
}
tex_params_pdf: axis_gaussian_builder
shape_params_pdf: axis_gaussian_builder
app_params_pdf: axis_gaussian_builder
n_pixels: 10000
colour: Grey // Alternatives: Grey,RGB,...
// Texture Sampler can be tri_raw, tri_edge...
tex_sampler: vapm_triangle_sampler<vxl_byte>
// tex_aligner can be None, AlignLinear1D, ...
tex_aligner: align_linear_1d
// shape_wts define how to compute relative scaling of shape & tex.
// shape_wts can be `EqualVar', `EqualEffect',...
shape_wts: EqualVar
// point_finder defines point finder used for ASM search
•point_finder: local_models
// tex_model defines type of model to represent texture statistics,
eg: pca, pca+haar1d
tex_model: pca
// Image Pyramid Builder can be gauss_byte, gauss_float, grad_float
...
pyr_builder: gauss_byte
points_pyr_builder: Same
max_im_pyr_levels: 5
```

```
// Levels of multi-res model to build :
min_level: 0
max_level: 4

// For ASM profile models :
g_in_len: 3
g_out_len: 3
g_width: 1
sample_int: 1
use_grad: 1
norm_grey: 1
restrict_points_to_parts: 0

default_image_pixel_size: 1
// Details of points : images
training_set:
{
  Bot53.pts : training_set/Bot53.tif

  training_set/Bot25.pts : training_set/Bot25.tif
  training_set/Bot42.pts : training_set/Bot42.tif
  training_set/Bot43.pts : training_set/Bot43.tif
  training_set/Bot46.pts : training_set/Bot46.tif
  training_set/Bot47.pts : training_set/Bot47.tif
  training_set/Bot87.pts : training_set/Bot87.tif
  training_set/Bot88.pts : training_set/Bot88.tif
  training_set/Bot110.pts : training_set/Bot110.tif
  training_set/Bot111.pts : training_set/Bot111.tif
  training_set/Bot168.pts : training_set/Bot168.tif
  training_set/Bot171.pts : training_set/Bot171.tif
  training_set/Bot174.pts : training_set/Bot174.tif
  training_set/Bot176.pts : training_set/Bot176.tif
  training_set/Bot178.pts : training_set/Bot178.tif
  training_set/Bot179.pts : training_set/Bot179.tif
  training_set/Bot211.pts : training_set/Bot211.tif
  training_set/Bot213.pts : training_set/Bot213.tif
  training_set/Bot229.pts : training_set/Bot229.tif
  training_set/Bot231.pts : training_set/Bot231.tif
  training_set/Top10.pts : training_set/Top10.tif

  training_set_extended/Bot4.pts : training_set_extended/Bot4.tif
  training_set_extended/Bot5.pts : training_set_extended/Bot5.tif
  training_set_extended/Bot14.pts : training_set_extended/Bot14.tif
  training_set_extended/Bot27.pts : training_set_extended/Bot27.tif
  training_set_extended/Bot34.pts : training_set_extended/Bot34.tif
  training_set_extended/Bot35.pts : training_set_extended/Bot35.tif
  training_set_extended/Bot36.pts : training_set_extended/Bot36.tif
  training_set_extended/Bot39.pts : training_set_extended/Bot39.tif
  training_set_extended/Bot46.pts : training_set_extended/Bot46.tif
  training_set_extended/Bot48.pts : training_set_extended/Bot48.tif
  training_set_extended/Bot52.pts : training_set_extended/Bot52.tif
  training_set_extended/Bot55.pts : training_set_extended/Bot55.tif
  training_set_extended/Bot64.pts : training_set_extended/Bot64.tif
  training_set_extended/Bot65.pts : training_set_extended/Bot65.tif
  training_set_extended/Bot71.pts : training_set_extended/Bot71.tif
  training_set_extended/Bot75.pts : training_set_extended/Bot75.tif
  training_set_extended/Bot76.pts : training_set_extended/Bot76.tif
  training_set_extended/Bot78.pts : training_set_extended/Bot78.tif
  training_set_extended/Bot95.pts : training_set_extended/Bot95.tif
  training_set_extended/Bot241.pts                                        :
training_set_extended/Bot241.tif
}
```

Appendix C Raw Data Table Example

This is an example of a raw data table used to input the results from each individual search test sample. It is reduced to show only 9 samples. The name of the points file from which the points were loaded is listed in the first column. The numbers 15, 20 and 40 are the test type representing the number of samples that were used in the training model. The numbers on the second bottom row are the totals of each column. The 9 represents the number of samples in the test set. Lastly, the percentages represent the percentage of samples successfully matched. The full raw data tables can be found on the accompanying CD.

	Test Type		
Sample	**15**	**20**	**40**
Bot1.pts	0	1	0
Bot1_a.pts	0	0	0
Bot206_b.pts	0	1	1
Bot218.pts	1	1	1
Bot224.pts	0	0	0
Bot228.pts	0	0	1
Bot228_a.pts	1	1	1
Bot228_b.pts	0	0	0
Bot233.pts	1	1	1
9	*3*	*5*	*5*
Matches:	**33.33%**	**55.56%**	**55.56%**

Appendix D Wrapper Program Code

The complete source code to the wrapper program can be found on the accompanying CD. This appendix contains a general description of the relevant methods used in preparing the segment data for the ASM Toolkit.

D.1 Overall Description

The wrapper works by first typing the name of a segment file from an initial segmentation program. Then the “load segments” button is clicked. This will load the data contained in this file into an array in the program. Once this is completed, the “start” button is clicked to start the program iterating through each segment in the array and transforming the points based on the input. The wrapper saves each points file as it calculates it, and at the end saves the overall SMD file.

D.2 actionPerformed Method

The actionPerformed method handles the button clicks. First the “load segments” button is clicked. This loads the segment file from the filename typed into the textfield and stores all the transformation data in an array. Then the start button is clicked that iterates through all the entries in that array. First it loads the default points file, then calls the startTranslation() method passing to it the transformation values in the array. Once this method finishes, the points are saved to a separate points file where the filename is based off the image filename.

```
public void actionPerformed(ActionEvent event)
{
String tempFile;
double xCoord, yCoord, length, orientation;

// If the load segment file button is clicked
if (event.getSource() == loadSegments) {
    // Load the segments from Lambert input
    loadSegmentFile();
}
// If the start transformation button is clicked
else if (event.getSource() == start) {

    // Transform the points for each segment in the input
    for (int i = 0; i < segments.size(); i++) {
        // Remove any spaces in the image filename
        tempFile = ((SegmentObject) segments.get(i)).getFile();
        imageToLoad = tempFile.replaceAll(" ", "");

        // Change the filename extension to .pts for the points file
        pointsFileToSave = imageToLoad.replaceAll(".tif", ".pts");
        pointsFileToSave = imageToLoad.replaceAll(".jpg", ".pts");

        // Load the default model to manipulate
        loadPoints(defaultPointsFile);

        // Get the segment data to translate the model from
        xCoord = ((SegmentObject) segments.get(i)).getXCoord();
        yCoord = ((SegmentObject) segments.get(i)).getYCoord();
        length = ((SegmentObject) segments.get(i)).getLength();
        orientation = ((SegmentObject) segments.get(i)).getOrientation();

        // Translate the model
        startTranslation(xCoord, yCoord, length, orientation);
```

```
        // Save the points file
        savePoints(pointsFileToSave);

        // Display progress in the cmd window
        System.out.println(Integer.toString(i + 1) +
                           " of " +
                           Integer.toString(segments.size()) +
                           ") " +
                           smd_entries.get(i) +
                           " saved."
                );
    }

    // Save the overall SMD file once all segments have been provessed
    saveSMD(SMD_filename);

    // Display file saved message in the cmd window.
    System.out.println(SMD_filename + " saved.");
}
}
```

D.3 startTranslation Method

This takes the transformation values and calculates the current centroid, then moves the centroid to location (0,0). It then rotates the points around the centroid and translates the centroid back to lie over the identified segment from the input. Lastly it scales the points using the length as the scaling value from the input.

```
public   void   startTranslation(double   xOffset,   double   yOffset,   double
                    segmentLength, double rotation)
{
double height, length, xCentroid, yCentroid;
double oldX, oldY;
double xDif = 0, yDif = 0;
int drawX, drawY;
double scaleNormalise;


// Calculate dimensions of the default points
length = points[12][0] - points[1][0];  // Length
height = points[22][1] - points[3][1];  // Height

// Calculate centroid of the default points
xCentroid = points[1][0] + length / 2.0;
yCentroid = points[3][1] + height / 2.0;


// Rotate points around the centroid  to match the loaded segment
// Iterate through each coordinate
for (int i = 0; i < numPoints; i++) {
    // Translate the point to the origin from the default model position
    points[i][0] -= xCentroid;
    points[i][1] -= yCentroid;

    oldX = points[i][0];
    oldY = points[i][1];

    // Calculate rotation
    points[i][0] = (oldX * Math.cos(-rotation)) -
                                (oldY * Math.sin(-rotation));
    points[i][1] = (oldX * Math.sin(-rotation)) +
```

```
                                (oldY * Math.cos(-rotation));

    // Translate the coordinates to the new segment's position
    points[i][0] += xOffset;
    points[i][1] += yOffset;
}


// Calculate new centroid location
xCentroid = points[1][0] + length / 2.0;
yCentroid = points[3][1] + height / 2.0;

// Convert segment length into a percentage of the length of current model
scaleNormalise = segmentLength / length * 100;

// Scale the points
for (int i = 0; i < numPoints; i++) {
    // Calculate the coordinates' offset to the centroid
    xDif = xCentroid - points[i][0];
    yDif = yCentroid - points[i][1];

    // Scale the coordinate by the percentage calculated previously
    points[i][0] = xCentroid - (xDif / 100 * scaleNormalise);
    points[i][1] = yCentroid - (yDif / 100 * scaleNormalise);
}
}
```

D.4 savePoints Method

This takes a filename as a parameter and saves the points contained in the points array to a file. This file has a small amount of additional data that needs to be saved as well in order for the file to be compatible with the ASM Toolkit.

```
public void savePoints(String filename)
{
PrintWriter Pwriter;
String inputLine, xTemp, yTemp, newFilename, fileInc;
StringBuffer fileContentsBuf = new StringBuffer("");
File pointsFile;

newFilename = filename;

// Set the fileInc to a default value (Used in nextLetter())
fileInc = " ";

try {
    // Create a pointer to the file that will be saved (test if it exists)
    pointsFile = new File(baseDir + pointsDir + specificPointsDir +
                          newFilename);

    // If the file does exist
    while(pointsFile.exists()) {
        // Increment a character
        fileInc = Character.toString(nextLetter(fileInc.charAt(0)));

        // Create a new filename with the character
        newFilename = filename.replaceAll(".pts", "_" + fileInc + ".pts");

        // Renew the pointer to test if this variation exists
        pointsFile = new File(baseDir + pointsDir + specificPointsDir +
                              newFilename);
    }

    // Create the unique file to write the points to
```

```
    Pwriter = new PrintWriter(new FileWriter(baseDir + pointsDir +
                                             specificPointsDir +
                                             newFilename, false), true);

    // Points file header
    fileContentsBuf.append("version: 1\r\nn_points: 29\r\n{\r\n");

    // Append the points to the buffer
    for (int i = 0; i < numPoints; i++) {
        // Convert the points to string and insert padding
        xTemp = Double.toString(points[i][0]) + "0000";
        yTemp = Double.toString(points[i][1]) + "0000";

        // Only include points to 3 dp precision
        xTemp = xTemp.substring(0, xTemp.indexOf(".")+5);
        yTemp = yTemp.substring(0, yTemp.indexOf(".")+5);

        // Save to string buffer
        fileContentsBuf.append(xTemp);
        fileContentsBuf.append(" ");
        fileContentsBuf.append(yTemp);
        fileContentsBuf.append("\r\n");
    }

    // Points file footer
    fileContentsBuf.append("}\r\n");

    // Write file
    Pwriter.write(fileContentsBuf.toString());

    Pwriter.close();

    // Save the filename to the smd_entries array for saving later
    // Format example: training_set/Bot25.pts : training_set/Bot25.tif
    smd_entries.add(specificPointsDir + newFilename + " : " +
                    specificImagesDir + imageToLoad);

} catch (IOException E) {System.out.print(E.toString());}

}
```

D.5 saveSMD Method

This saves the shape model data (SMD) file. First it writes all the required parameters that define file locations. Then it writes the name of each points file and its associated image file as a list.

```
public void saveSMD(String filename)
{
PrintWriter Pwriter;
String inputLine, xTemp, yTemp;
StringBuffer fileContentsBuf = new StringBuffer("");

try {

    Pwriter = new PrintWriter(new FileWriter(baseDir + modelsDir +
                                             filename, false), true);

    // SMD file header
    fileContentsBuf = getSMD_header();

    // Append the testing entries to the buffer
    for (int i = 0; i < smd_entries.size(); i++) {
        fileContentsBuf.append(smd_entries.get(i) + "\r\n");
```

```
    }

    // SMD file footer
    fileContentsBuf.append("}\r\n");

    // Write file
    Pwriter.write(fileContentsBuf.toString());

    Pwriter.close();

} catch (IOException E) {System.out.print(E.toString());}

}
```

D.6 loadSegmentFile Method

This loads the segment file from the segmentation program (Lambert 2005). Each line represents a segment that is loaded into the segments array alont with the segment transformation data.

```
public void loadSegmentFile(){
String inputLine, filename;
StringTokenizer tokens;
SegmentObject tempSegment;


filename = segmentFileName.getText();


try {
    BufferedReader infile = new BufferedReader(new FileReader(baseDir +
                                                pointsDir + filename));

    // Get first line
    inputLine = infile.readLine();

    // While the line is not null
    while (inputLine != null) {

        tokens = new StringTokenizer(inputLine, "|");

        // Read the segment data from the line
        while (tokens.hasMoreTokens()) {
            // Add data to a segment object
            tempSegment     =     new     SegmentObject(tokens.nextToken(),
                                Double.parseDouble(tokens.nextToken()),
                                Double.parseDouble(tokens.nextToken()),
                                Double.parseDouble(tokens.nextToken()),
                                Double.parseDouble(tokens.nextToken()));

            // Add the segment object to an array list
            segments.add(tempSegment);
        }

        // Get next line
        inputLine = infile.readLine();
    }

    infile.close();

} catch (FileNotFoundException ex) {
    System.out.println("Segments file not found: " + ex.toString());
} catch (IOException ex) {
    System.out.println("Problem opening data file: " + ex.toString());
```

```
}
}
```

D.7 loadPoints Method

This loads the coordinates from a points file and stores them in the points array to manipulate for each segment.

```
public void loadPoints(String filename)
{
String inputLine;
StringBuffer fileContentsBuf = new StringBuffer("");
boolean readingPoints = false;
StringTokenizer tokens;
int i = 0;

try {
    BufferedReader infile = new BufferedReader(new FileReader(baseDir +
                                                pointsDir + filename));

    // Get first line
    inputLine = infile.readLine();

    // While the line is not null
    while (inputLine != null) {
        // If the current line is a closing brace, stop reading
        if (inputLine.equals("}")) readingPoints = false;

        // If still reading coordinates
        if(readingPoints) {
            tokens = new StringTokenizer(inputLine);

            // Read the (x,y) coordinates from the current line
            while (tokens.hasMoreTokens()) {
                points[i][0] = Double.parseDouble(tokens.nextToken());
                points[i][1] = Double.parseDouble(tokens.nextToken());
                i++;
            }

        }

        // If the current line is an opening brace,
        // start reading the next line
        if (inputLine.equals("{")) readingPoints = true;

        // Get next line
        inputLine = infile.readLine();
    }
    infile.close();
} catch (FileNotFoundException ex) {
    System.out.println("Points file not found: " + ex.toString());
} catch (IOException ex) {
    System.out.println("Problem opening data file: " + ex.toString());
}
}
```

D.8 Other Relevant Methods

There is another class called segmentObject used to store the segment data. There is an array within the wrapper of type segmentObject where each entry contains an object with the segment data. This was simply used as a convenient way to store the parameters used in transforming the model.